Un cluster de calcul hybride pour les applications de vision temps réel

Joël FALCOU[1], Jocelyn SÉROT[1], Thierry CHATEAU[1], Frédéric JURIE[2]

[1]LASMEA, UMR6602 UBP/CNRS, Campus des Cézeaux, 63177 Aubière, France.

[2]GRAVIR, INRIA/CNRS, 55 avenue de l'Europe, Montbonnot 38330, France

Joel.Falcou@lasmea.univ-bpclermont.fr Jocelyn.Serot,@lasmea.univ-bpclermont.fr
Thierry.Chateau@lasmea.univ-bpclermont.fr, Frederic.Jurie@inrialpes.fr

Résumé – On décrit l'implantation d'une chaîne de traitement et de reconstruction de points 3D en temps réel sur un cluster de PowerPC G5 biprocesseurs associant trois niveaux de parallélisme (SIMD grain fin, SMP et MIMD à passage de messages) et un dispositif dédié de diffusion de flots vidéo. Nous montrons que ce type d'architecture, permettant d'exploiter plusieurs formes de parallélisme, répond aux exigences des applications de vision temps réel en alliant puissance de calcul, facilité de programmation et extensibilité. Des résultats préliminaires sont fournis et discutés.

Abstract – We describe the implementation of an application performing real-time reconstruction of 3D points from stereo video streams on a cluster of G5 processors. The goal is to show that the combined use of a dedicated broadcasting mechanism for input streams and a three-level parallel programming model involving SIMD,SMP and MIMD/MP parallelism makes this type of platform a suitable solution for the implementation of real-time vision applications involving complex algorithms and requiring large amounts of computing power. Preliminary results are shown and discussed.

1 Introduction

La vision artificielle est une discipline exigeant de fortes puissances de calcul, particulièrement lorsqu'il s'agit de traiter des flots d'images en opérant à la volée des capteurs. Pour ces applications, les traitements doivent se faire en temps réel pour permettre aux systèmes d'interagir avec leur environnement (applications de robotique mobile, interfaces homme machine, etc.).

Conscient à la fois de l'importance de ce type d'applications et des contraintes qui y sont liées, les chercheurs du domaine ont utilisé diverses stratégies pour accroître la réactivité de tels systèmes : dégradation des algorithmes, utilisation de matériels spécifiques pour les traitements de bas niveau (FPGA, processeurs des cartes graphiques, etc.) ou d'architectures distribuées. Ces stratégies se sont parfois montrées efficaces en raison des propriétés spécifiques des images et des structures des algorithmes associés. Cependant, la progression constante et rapide de la puissance de calcul des ordinateurs personnels rend l'utilisation de solutions spécifiques de moins en moins attractive : les délais supplémentaires liés au temps de développement de ces solutions et de leurs logiciels, leur coût important, plaident en effet désormais en faveur de l'utilisation d'architectures fondées sur l'usage systématique de composants standards.

Dans le domaine du calcul scientifique hautes performances (simulation, modélisation, :.) cette classe de solution a vu son audience exploser ces dix dernières années avec la généralisation des clusters (grappes, fermes) de calcul bâties à partir d'ordinateurs individuels. Mais ce type de solution n'a pratiquement jamais été mis en œuvre dans le contexte d'applications de vision complexes devant opérer à la volée de capteurs. Le travail décrit ici vise donc à évaluer cette opportunité en proposant une architecture de type cluster dédiée aux applications de vision temps-réel. Nous décrivons l'architecture matérielle d'une telle plate-forme – en justifiant les choix technologiques effectués à partir des exigences du domaine d'application et de l'état de l'art – puis l'infrastructure logicielle associée, en insistant sur les aspects programmabilité. La validité de l'approche est démontrée en décrivant l'implantation d'une application de reconstruction de points d'intérêt 3D à partir d'un flot d'images stéréo et les performances obtenues avec une première implémentation.

2 Architecture du cluster

2.1 Architecture matérielle

L'architecture matérielle du cluster est présentée sur la figure 1 et comprend 14 noeuds de calcul. Chacun de ces noeuds est composé d'un PowerPC G5 biprocesseur [1] cadencé à 2 GHz et équipé de 1Go de mémoire. Les noeuds sont interconnectés par un réseau Ethernet Gigabit et reçoivent un flux vidéo stéréoscopique en provenance d'une paire de caméras numériques via un bus *FireWire IEEE1394a* [2]. Cette structure permet une diffusion simultanée et synchronisée des images sur tous les noeuds, éliminant ainsi le goulet d'étranglement classique consistant à redistribuer à l'ensemble des noeuds les images de travail via le réseau Ethernet. Dans notre configuration, la bande passante Ethernet reste disponible pour le transfert de message entre les noeuds. Le choix du processeur G5 est motivé par les points suivants :

– **La présence de l'extension multimédia Altivec [7].** Cette extension peut fournir des accélérations de l'ordre de 2 à 12 pour de nombreuses classes d'algorithmes bas et moyen niveau. [5]. Sans cette extension, la taille du

1. Apple XServe Cluster Node
2. Fonctionnant en mode isochrone à 400Mo/s

cluster nécessaire à l'obtention de l'accélération voulue (typiquement 50) aurait été prohibitive tant en termes de coût que de consommation.

– **La gestion native de l'Ethernet Gigabit**.
L'Ethernet Gigabit offre un des meilleurs ratio coût/performance. Des solutions plus onéreuses comme MyriNet ou InfiniBand ont été envisagées mais leur coût élevé aurait diminué sensiblement le nombre de noeud du cluster.

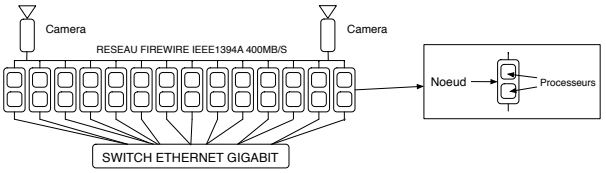

FIG. 1: *Architecture du cluster*

L'architecture proposée est suffisamment ouverte pour lui permettre de bénéficier des derniers progrès technologiques tout en supportant des modèles et des outils de programmation standard. En outre, l'utilisation de ces outils et modèles standards permet de rendre le cluster évolutif en permettant une mise à jour matérielle ponctuelle des noeuds.

2.2 Architecture logicielle

L'architecture logicielle du cluster est structurées selon trois niveaux.

Au niveau le plus bas, un **pilote *Firewire***[6] gère le transfert des images des caméras vers la mémoire du processeur en utilisant un canal DMA dédié. Pour l'utilisateur, les appels à ce pilote sont encapsulés dans une bibliothèque dédiée qui fournit des méthodes pour configurer les caméras, obtenir un pointeur vers l'image qui vient d'être reçue et synchroniser l'acquisition sur le cluster (sec. 3.3) Le second niveau est dédié à la mise en oeuvre des trois niveaux de parallélisme offerts par le cluster : un parallélisme de type SIMD à grain fin au sein de chaque processeur (Altivec), un parallélisme de type SMP entre les deux processeurs de chaque noeud et un parallélisme de type SPMD à passage de message entre les processeurs des différents noeuds. Le premier type de parallélisme est directement exploitable via l'interface C de l'extension AltiVec ou via EVE [5], une bibliothèque de vectorisation optimisée pour cette extension. Le parallélisme SMP nécessite la définition explicite des processus légers à exécuter sur chaque noeud via la bibliothèque PTHREAD. Enfin, l'application est décomposée en groupe de processus SPMD communiquant entre eux grâce à MPI. L'utilisation explicite de PTHREAD est due au fait que la gestion des pilotes Firewire n'est pas prévue pour être appelée de manière transparente par deux processeurs. Chaque noeud est vu comme un seul noeud Firewire par le pilote et comme deux noeuds par MPI. Cette dualité provoque des conflits de partage de ressources que MPI ne peut gérer facilement.

Enfin, un framework développé en C++ fournit, via l'utilisation de patrons prédéfinis – squelettes au sens de [8] – des fonctions de gestion des entrés/sorties vidéos et des mécanismes permettant des modifications dynamiques des modules algorithmiques de l'application, de développer et de mettre au point rapidement des applications pour le cluster. Créer une nouvelle application se résume alors à écrire des modules algorithmiques indépendants et à définir une stratégie d'exécution de ces modules. Ces stratégies peuvent êtres stockés sous forme de simple fichier texte et activés de manière dynamique au cours de l'exécution de l'application, soit de manière interactive ou pré-programmée. Le code présenté sur la figure 2 est un exemple simple d'utilisation de notre architecture logicielle. Dans cet exemple, nous construisons une application qui va traiter une image en provenances d'une caméra numérique et y appliquer un seuillage. Dans ce but, nous utilisons un squelette de parallèlisation classique : le *Scatter-Compute-Merge*.[3]

```
struct Data : public Singleton<Data> {
  REGISTER_ELEMENT(Frame, in);
  REGISTER_ELEMENT(Frame, out);
};

struct Seuil : public Task {
  virtual void run( int nodeID ) {
   Data::out() = where(Data::in() > 127, 255, 0);
  }
};

int main(int argc, const char** argv )
{
  Camera input( 30, res640x480xmono );
  Cluster c(argc,argv, MPI_COMM_WORLD);

  Task* a = SCM<RowSplit,Seuil,RowMerge>(0,5);
  c.assignTask(a);

  input >> Data::in();
  c.run(Data::Instance());
  return 0;
}
```

FIG. 2: *Une application de seuillage parallélisée sur notre cluster.*

La première partie de ce code consiste à définir une structure `Data` qui sera utilisée par tous les modules algorithmiques pour stocker les données de l'application. Dans cet exemple, `Data` contient deux instances de la classe `Frame` qui encapsule le concept d'image vidéo. Nous définissons ensuite la classe `Seuil` qui sera notre module de traitement. Cette classe fournit une méthode `run` qui contient le traitement proprement dit. Ici, le seuillage est effectué via la fonction `where` fournie par EVE . Cette fonction est une version SIMD de la construction `if then else`. L'application proprement dite définie une instance de la classe `Camera` ainsi que de la classe `Cluster`. Nous définissons alors la tache à effectuer sur ce `Cluster` en utilisant un des squelettes parallèles prédéfinis. `SCM` est un squelette de parallélisation qui va découper l'image d'entrée en bande, les diffuser sur un nombre fixe de noeuds de calcul, appliquer une opération et récupérer les résultats de ces calculs afin de reconstruire l'image de sortie. La fonction `SCM` prend en paramètres les classes définissant les stratégies de découpages et de fusion (`RowSplit` et `RowMerge`), celle définissant les opérations à effectuer (ici `Seuil`) et le nombre de noeuds de calculs (ici 5). Une fois cette tache définie, l'application commence à acquérir des images de la caméra et à les traiter via le `Cluster`.

3. Pour un souci de brièveté, nous avons volontairement omis les directives d'inclusions et les déclarations de *namespace*.

3 Application

Nous avons évalué cette plate-forme à partir d'applications vidéos temps réel pour démontrer que son architecture matérielle (double bus : Ethernet pour les communications et *Firewire* pour la diffusion vidéo) et logicielle (modèle de programmation parallèle à trois niveaux : SIMD, SMP, MP) convient aux besoins de cette classe d'applications tant en termes de performance que de flexibilité.

3.1 Description de l'algorithme

Nous avons implanté les premières étapes d'une chaîne de reconstruction et suivi 3D utilisant la vision stéréoscopique. L'application décrite ici génère une ensemble de points 3D qui seront par la suite traités par un algorithme d'interprétation et de suivi que nous ne décrivons pas ici. Cet algorithme peut se décomposer en quatre étapes :

1. **Rectification de l'image (RECTIF)** : Cette étape consiste à appliquer une homographie aux images d'entrée en mettant en correspondances les droites épipolaires et les lignes de pixels de l'image[9]. Cette étape permet ensuite de simplifier la recherche des correspondances en limitant cette recherche à une ligne de pixels.
2. **Détection de point d'intérêts (DETECT)** : Un détecteur de coins de Harris et Stephen [4] est utilisé pour extraire des points d'intérêts qui seront autant de germes pour l'étape de mise en correspondance
3. **Mise en correspondance (MATCH)** : Chaque point d'intérêt de l'image gauche (resp. de l'image droite) est mis en correspondance avec un point d'intérêt de l'image droite (resp. gauche) en utilisant un recherche du maximum de vraisemblance par corrélation centrée réduite.
4. **Reconstruction 3D (BUILD)** : On effectue une triangulation [10] des paires de points 2D correctement mis en correspondance.

Nous avons choisi cet algorithme pour trois raisons : le grand nombre d'opérations par image, la diversité de ces opérations et le fait qu'il soit à la base de nombreuses autres applications comme la navigation de robot mobile ou l'interface homme/machine. Un exemple de reconstruction est donné sur la figure 3.

3.2 Stratégie de parallélisation

L'ensemble des quatre étapes décrites ci-dessus est exécuté en découpant l'image source en cinquante bandes horizontales. Chacune de ces bandes est ensuite distribuée à un processeur[4]. Comme le nombre de points d'intérêts dans chaque bande ne peut être prédit et donc que la charge de calcul associée peut fluctuer de manière significative d'une bande à l'autre, le squelette `SCM` présenté dans l'exemple 2 est inadapté. Un autre squelette de parallèlisation classique, `Farm`, fournit une méthode simple d'équilibrage de charge. Dans ce squelette, un noeud maître distribue de manière dynamique des bandes d'image à une réserve d'esclaves. Dès qu'un des esclaves à terminé d'effectuer sa tâche sur une bande, le maître lui en fournit une nouvelle. Au final, les ensembles de points 3D en provenance de tous les esclaves sont fusionnés et rapatrié sur le processeur maître. PTHREAD permet de distribuer sur les processeurs de chaque noeud une bande provenant de l'image droite et la bande correspondante de l'image gauche. A un moment donné, chaque noeud traite ainsi des bandes d'images provenant bien d'une même portion des images initiales. Les étapes de détection, de mise en correspondance et de reconstruction mettent en jeu des opérations iconiques ou géométriques régulières (convolution, produit scalaire) Elles sont donc vectorisées par EVE [5] pour profiter de l'accélération fournie par l'unité SIMD du PowerPC G5.

4. Ce qui est possible grâce à l'étape de rectification qui force les points d'intérêts à s'aligner sur les lignes de pixels de l'image.

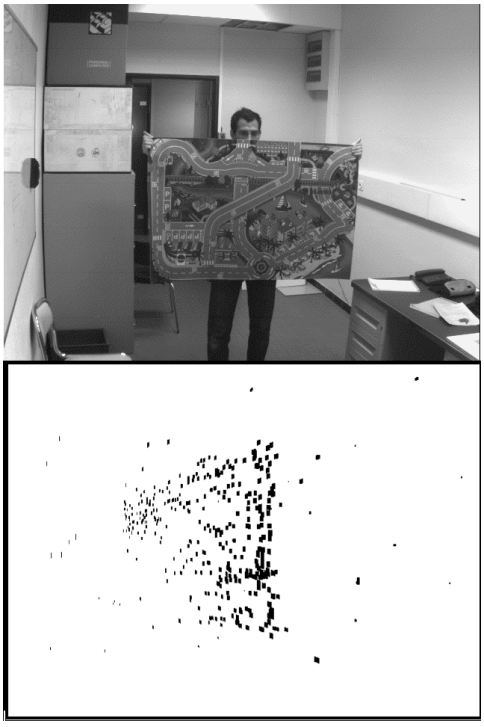

FIG. 3: *Résultat de la reconstruction 3D*

3.3 Synchronisation des Cameras

Pour assurer une grande précision dans la reconstruction, l'algorithme doit être appliqué à des paires d'images décrivant le même événement. Si les images droites et gauches de la paire sont désynchronisées, la mise en correspondance et la reconstruction seraient de mauvaise qualité. En pratique, cela implique que l'acquisition des images droite et gauche sur un noeud doit être synchronisé. De même, la diffusion des images sur tous les noeuds du cluster doit être synchronisé. Cette double synchronisation est réalisée comme suit :

- **Synchronisation intra-noeud** : le pilote Firewire CFOX est capable par un système de datation des images et de triple-buffering de fournir des images synchronisées pour un nombre arbitraire de caméras. Cette synchronisation est totalement transparente pour l'utilisateur qui n'a pas à se soucier de connaître la datation des images. L'appel de la méthode d'acquisition gére ces problèmes de manière transparente.
- **Synchronisation inter-noeud** : En utilisant une synchronisation par passage de messages (via `MPI_Barrier`),

il est possible de forcer les noeuds à s'attendre et à se synchroniser pour acquérir la nouvelle image depuis le bus Firewire.

4 Résultats

Des résultats préliminaires, obtenus sur des images 640×480 ×8bits sont donnés dans le tableau 1. Les temps de calculs de chaque étape ainsi que le temps d'exécution total de l'algorithme sont donnés pour un nombre de processeurs variable en ms[5]. La première colonne donne pour référence le temps d'exécution pour une version séquentielle de l'algorithme sur un seul processeur.

Etape	n=1	n=2	n=4	n=8	n=16	n=26
RECTIF	246	139, 1	70, 5	36, 1	19, 5	13, 1
DETECT	262	80, 1	40, 5	20, 6	11, 2	7, 1
MATCH	304, 2	180	91, 5	47, 4	22, 4	13, 8
BUILD	180	100	53	27, 5	18, 2	12, 0
TOTAL	992, 2	479, 2	244, 6	122, 6	68, 2	42, 9

TAB. 1: *Résultats préliminaires de recon-3D*

La latence minimale est de 42.9ms, obtenue avec 26 processeurs, soit une cadence de 24 images par seconde (les caméras fonctionnant à 30 images/secondes). L'accélération est alors de 23.1 (soit une efficacité de 88%[6]). Deux facteurs expliquent ces résultats encourageants. Premièrement, grâce à la diffusion automatique des images sur le bus Firewire, l'application montre un ratio calcul / communication très élevé (entre 0.90 et 0.92). Deuxièmement, la parallélisation grain fin SIMD est exploité de manière efficace dans les étapes DETECT et MATCH. Sans AltiVec, la cadence tombe à 19 images/secondes, démontrant que la couche SIMD fournit une accélération non négligeable. Signalons que ces résultats préliminaires ont été obtenus avec une implantation naïve des algorithmes. Un grand nombre de paramètres et de modèles peuvent êtres affinés pour permettre une utilisation plus intensive de EVE et donc un gain final plus important. Le nombre d'esclaves dans le squelette `Farm`, la taille et le nombre de bande sont autant de paramètres dont les variations ont potentiellement un impact sur les performances de l'application.

5 Travaux en rapport

Pour la reconstruction 3D temps réel, une solution classique consiste à reconstruire l'enveloppe visuelle d'un objet à partir de silhouettes en utilisant N caméras chacune connectées à un noeud. Les caméras sont synchronisées afin d'assurer que chaque paire PC/Camera traite une vue du même événement. La parallélisation de ces algorithmes utilise un simple pipeline ou chaque étape de l'algorithme est exécutés en parallèle sur chaque noeud. Wu et Matsuyama [1] ou Franco et al. [2] aboutissent ainsi à des résultats temps réel avec un nombre restreint de cameras et de noeuds. Ces approches différent de celle présentée ici tant au niveau des stratégies de parallélisation que de l'utilisation des caméras.

5. Les deux processeurs dédiés à l'affichage et à la synchronisation globale ne sont pas pris en compte.

6. 82% en tenant compte des deux processeurs de contrôle et d'affichage.

Yoshimoto et Arita[3] décrivent quant à eux un cluster de PC utilisant un réseau Firewire pour le traitement d'images en temps réel. Ce cluster utilise un bus Firewire IEEE-1394a à la fois pour transmettre les images à traités et les communications inter-noeuds. Des applications temps réels sont montrées mais avec cette architecture, l'utilisation du bus devient rapidement un goulet d'étranglement, ce qui limite sérieusement les performances et l'extensibilité du cluster.

6 Conclusion

Cet article montre qu'un cluster doté d'un mécanisme de diffusion vidéo performant et offrant plusieurs niveaux de parallélisme permet de satisfaire les contraintes temporelles d'applications de vision temps réels, ce que nous avons démontré en développant une application de reconstruction 3D à partir d'un flux vidéo stéréoscopique. À notre connaissance, il s'agit de la première utilisation d'une telle architecture pour permettre l'exécution d'une application temps réel de cette complexité. Ces résultats restent néanmoins préliminaires. À court terme, la finalisation de la chaîne de reconstruction ainsi que le développement d'une application de suivi 3D est prévue.

Références

[1] Wu X., Matsuyama T. Real-time active 3d shape reconstruction for 3d video. In Proc. of 3rd International Symposium on Image and Signal Processing and Analysis, pages 186-191, 2003.

[2] Franco J.-S., Menier C. and Boyer E. A Distributed Approach for Real-Time 3D Modeling.CVPR Workshop on Real-Time 3D Sensors and their Applications, 2004

[3] Yoshimoto H. and al. Real-Time Image Processing on IEEE1394-based PC Cluster. In 15th International Parallel and Distributed Processing Symposium, 2001

[4] Harris C. and Stephens M. A combined corner and edge detector. In 4th Alvey Vision Conference, 1988, pp. 189-192.

[5] Falcou, J., Serot, J - E.V.E., An Object Oriented SIMD Library. In Practical Aspects of High-level Parallel Programming,ICCS 2004, pp. 323-330(3)

[6] Falcou, J. - CFOX - A Firewire Camera Driver for OS X `http://wwwlasmea.univ-bpclermont.fr/Personel/Joel.Falcou/software/cfox`

[7] Ollman I. AltiVec Velocity Engine Tutorial. `http://www.simdtech.org/altivec`. Mars 2001.

[8] Cole M. - Algorithmic Skeletons. In Research Directions in Parallel Functional Programming, Springer-Verlag, 1999.

[9] Fusiello A.,Trucco E., Verri A. - A compact algorithm for rectification of stereo pairs. In Machine Vision and Application, vol. 12, no. 1, pp. 16–22, 2000

[10] Hartley R. I. and Zisserman A. - Multiple View Geometry in Computer Vision. Cambridge University Press - ISBN 0521623049, 2000.